ISEL
Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

2º Trabalho Prático

Trabalho realizado por:

Nome: António Paulino Nº 50512

Nome: Bernardo Pereira Nº 50493

Turma: LEIC23D

Docente: João Patriarca

Arquitetura de Computadores

2022 / 2023 verão

21 de Abril de 2022

Índice

1 Análise da microarquitetura

1. A microarquitetura do processador em estudo é do tipo Harvard, já que existem dois barramentos distintos, um para os dados, e outro para o código. Para além disso, as instruções são executadas com apenas um ciclo de clock já que os dados e o código das instruções podem ser acedidos ao mesmo tempo.

2. A função do bloco EXT (Extend) tem a função de extensão do sinal, sendo que o tipo de extensão depende do sinal SE (Signal Extend). Isto é, quando SE -> 1, EXT completa o sinal de entrada (6bits) com mais 2 bits iguais ao de maior peso EX: 100011 -> 11100011 (extensão de sinal). Quando SE -> 0, EXT completa o sinal com 2 bits iguais a 0 EX: 100011 -> 00100011 (Zero fill). No âmbito da instrução bne, os bits são estendidos com SE -> 1, isto porque é preciso fazer uma extensão de sinal ao valor label, já que no âmbito desta instrução o offset adicionado a PC pode ser interpretado como um valor negativo. No âmbito da instrução mov, os bits são estendidos com SE -> 0, já que nesta instrução a constante a 3 bits codificada na instrução é sempre interpretada como um natural.

3. O modo de endereçamento associado à instrução bne label é direto, já que o endereço é codificado da instrução. O modo de endereçamento associado à instrução b rn é indireto, já que o endereço é especificado através de um registo. A vantagem de endereçamento direto em relação a endereçamento indireto é de que é mais simples e eficiente, já que o endereço é codificado diretamente na instrução e não necessita o uso de registos. Uma das desvantagens em relação a endereçamento indireto pode ser o facto de que o endereço não pode ser alterado enquanto o programa corre como não usa registos, caberia ao programador modificar a instrução.

2 Codificação de instruções

1.

Intruções	B			A			Opcode		
Bits	8	7	6	5	4	3	2	1	0
add rd, rn	rn			rd			0	0	0
b rn	-	-	-	rn			0	1	0
bne label	label						0	0	1
cmp rn, rm	rm			rn			1	0	1
ldr rd, [rn]	rn			rd			1	1	1
mov rd,#imm3	#imm3			rd			1	0	0
push rn	-	-	-	rn			1	1	0
str rd, [rn]	rn			rd			0	1	1

Os registos de destino (das intruções que têm registo destino) foram codificados em A, já que o registo B precisa de ser utilizado como endereço no âmbito das instruções ldr e str devido ao caminho de dados desta microarquitetura. Para além disso, como o sinal Op da ALU é um sinal que vem diretamente da instrução, esse sinal foi codificado nos bits 1 e 0 do Opcode (relevante apenas nas instruções que necessitam da ALU, add, cmp, ldr…)

2.

Intruções	opcode		
Bits	2	1	0
add rd, rn	1	0	0
b rn	1	1	0
bne label	1	0	1
cmp rn, rm	0	0	1
ldr rd, [rn]	0	1	1
mov rd,#imm3	0	0	0
push rn	0	1	0
str rd, [rn]	1	1	1

Tendo em conta o funcionamento da ULA, o sinal de controlo Op da ULA serão os bits 1 e 0 do opcode (00 para add, 01 para cmp…) isto porque no add se realiza uma adição, no cmp uma subtração, em push é realizada uma decrementação ao registo SP (B – 1), e nas operações str e ldr é utilizado o registo B como endereço.

3 Projeto do descodificador de instruções

1.

	opcode			Z	SO	SI	SS	SE	SD	ER	EP	nRD	nWR
add rd, rn	0	0	0	-	0	0	0	-	1	1	1	1	1
bne label	0	0	1	0	1	0	-	Opcode(0)	0	0	0	1	1
bne label	0	0	1	1	0	0	-	-	-	0	0	1	1
b rn	0	1	0	-	-	1	-	-	-	0	0	1	1
str rd, [rn]	0	1	1	-	0	0	0	-	-	0	0	1	0
mov rd, #imm3	1	0	0	-	0	0	0	Opcode(0)	0	1	0	1	1
cmp rn, rm	1	0	1	-	0	0	0	-	-	0	1	1	1
push rn	1	1	0	-	0	0	1	-	1	1	0	1	0
ldr rd, [rn]	1	1	1	-	0	0	0	-	2	1	0	0	1

As instruções foram descodificadas de acordo com o caminho de dados desta microarquitetura, e a descrição de cada instrução no ponto 2 do enunciado.

2.

	IC2	IC1	IC0	Z	SO	SI	SS	SE	SD		ER	EP	nRD	nWR
	A3	A2	A1	A0	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
add rd, rn	0	0	0	-	0	0	0	-	0	1	1	1	1	1
bne label	0	0	1	0	1	0	-	1	0	0	0	0	1	1
bne label	0	0	1	1	0	0	-	-	-	-	0	0	1	1
b rn	0	1	0	-	-	1	-	-	-	-	0	0	1	1
str rd, [rn]	0	1	1	-	0	0	0	-	-	-	0	0	1	0
mov rd, #imm3	1	0	0	-	0	0	0	0	0	0	1	0	1	1
cmp rn, rm	1	0	1	-	0	0	0	-	-	-	0	1	1	1
push rn	1	1	0	-	0	0	1	-	0	1	1	0	1	0
ldr rd, [rn]	1	1	1	-	0	0	0	-	1	0	1	0	0	1

3. A ROM tem 2^4 = 16, endereços, já que tem 4 entradas, e cada um desses endereços guarda 10 bits(sinais de controlo). Então, a capacidade da memória rom é 16x10 = 160 bits.

4 Codificação de programas em linguagem maquina

1.

A função deste troço de código é de somar os primeiros 4 (valor em r2) elementos de um array em memória e guardar o valor na quinta posição (posição diretamente a seguir à do último elemento adicionado). Para esta operação, o programa começa por inicializar o apontador r0 com o valor 0, o registo das somas consecutivas r1 a 0, e o valor de r2 com a constante 4, que indica o índice do último elemento do array. De seguida entra num loop que carrega em r3 o valor do elemento do array na posição do endereço r0 (inicialmente 0, portanto carrega o primeiro elemento do array), e de seguida adiciona esse valor r3 a r1 para guardar a soma. Depois o apontador é incrementado por 1, e são feitas as somas consecutivas dos elementos do array a r1, o loop repete-se até o valor r0 ser igual ao valor de r2, ou seja, até ao apontador chegar ao último elemento do array. Quando chega ao último elemento do array, o valor da soma consecutiva dos primeiros 4 elementos é guardado na quinta posição do array. No final do programa, é carregado em r5 o valor 6 e r5 é incrementado por 6 o que resulta no valor 12 em r5. De seguida faz-se um branch para o valor r5. Na arquitetura deste processador, cada instrução ocupa 9 bits em memória e como cada endereço na memória de instruções guarda 9 bits, pode-se concluir que em cada instrução o endereço em memória é incrementado por 1. Dado que existem 13 instruções e a primeira tem endereço 0x0000, a instrução final (branch r5) terá o endereço 0x000C (ou 12 em decimal). Então, ao fazer um branch para a instrução com o valor 12 em memória, o programa estará sempre a voltar à instrução branch num ciclo infinito, o que finaliza a operação.

2.

Instrução	Endereço	Código Máquina
mov r0, #0	0x0000	0x0004
mov r1, #0	0x0001	0x000C
mov r2, #4	0x0002	0x0114
ldr r3, [r0]	0x0003	0x001F
add r1, r3	0x0004	0x00C8
mov r4, #1	0x0005	0x0064
add r0, r4	0x0006	0x0100
cmp r0, r2	0x0007	0x0085
bne loop	0x0008	0x01B9
str r1, [r2]	0x0009	0x008B
mov r5, #6	0x000A	0x01AC
add r5, r5	0x000B	0x0168
b r5	0x000C	0x01EA

Na instrução bne label, o offset é calculado através da diferença entre o endereço destino e o endereço atual, (neste caso 000011 - 001100 = 110111). As restantes instruções são codificadas de acordo com a codificação de instruções realizada no ponto 2.1 deste relatório. Os endereços são atualizados de acordo com o raciocínio da resposta anterior.